Homework Batch I: Matrix Multiplication

Gonzato Andrea

March 20, 2021

All the code of my work related to this homework can be found in this file matrix.py [1].

Exercise 1

The implementation of the function *strassen_matrix_mult()* can be found in the file matrix.py [1]

Exercise 2

The implementation of the general case can be found in the function *matrix_mult_generalization()*. Which can perform the multiplication of two matrices of any size $(n \times p)(p \times m)$.

The idea for this function is to up scale the dimensions of the couple of matrices to the closest power of 2. To fills each of those bigger matrices we put the original matrix in the top left corner and fills the rest of the free space whit zeros. Then with these new matrices we can compute the Strassen algorithm and at the end we just need to down scale the matrix obtain after the multiplication.

Follow the demonstration of the time complexity of the algorithm:
Let $g(n, m, p) := next\_power\_of\_2(max(n, m, p))$

- IF (n > m and n > p)
  Let $f(n) := next\_power\_of\_2(n)$
  $g(n, m, p) = f(n) \in \Theta(n)$
  The time complexity $T(n, m, p)$ of the algorithm can be express as:
  $T(n, m, p) = 2 \cdot \Theta(f(n)^2) + T_s\left(f(n)^{\log_2 7}\right)$
  Where the first term is due to the creation of the two bigger matrices and the second term is the time complexity of the execution of the Strassen algorithm.
  $T(n, m, p) = \Theta(n^2) + \Theta\left(n^{\log_2 7}\right) = \Theta\left(n^{\log_2 7}\right)$
  An analog demonstration can be given for the other two cases
- IF (m > n and m > p)
  ...
- IF (p > n and p > m)
  ...

In conclusion we can say that the time complexity of this algorithm is: $\Theta\left(k^{\log_2 7}\right)$
Where $k := max(n, m, p)$

Exercise 3

3.1 Matrices reuse

To improve the implementation and use less auxiliary matrices I implemented a new version of the Strassen algorithm where the result of each of the seven recursion is immediately added to the correct position of the result matrix.

The code of this implementation can be found in the function call *strassen_matrix_mult_improved()* in which the matrix S is reuse for each recursion call. By reusing this allocation we have less auxiliary matrices.

3.2 Null check

To further improve the implementation of all the multiplication functions. I decided to make the algorithms faster by using less operation and less matrices when is try to multiply two matrices and one of those is a null matrix which elements are all zeros.

To implement this improvement I changed all the multiplication functions by adding an optional boolean parameter (*check_null_matrix*). When this parameter is true, before run the original algorithm, I check if one of the two matrices is null. If this is the case the multiplication is trivial and I can give immediately the result.

3.3 Test

To evaluate the effect of those improvements I did a test. I generated a sequence of two random matrices of different size $(n \times p)(p \times m)$ and evaluated the time, in seconds, for the multiplication using all the four different algorithm. The standard is the generalization version of the Strassen Algorithm. The fourth version combine the previous two improvement together to get a better one. Below there is a table of the results of the test obtained using my computer.

n	p	m	standard	matrices reuse	null check	both improvements
3	1	5	0.001	0.000	0.001	0.000
5	3	9	0.010	0.002	0.004	0.002
9	5	17	0.081	0.009	0.015	0.005
17	11	33	0.443	0.066	0.079	0.023
33	21	65	2.970	0.484	0.466	0.126
65	43	129	21.286	3.614	3.039	0.698
129	85	257	148.563	24.601	18.708	3.974

Exercise 4

Using the "Matrices reuse" version the minimum auxiliary space require to compute the multiplication is:
$W(n) = 3 \cdot n^2 + 4 \cdot (n/2)^2 + W(n/2) = O(n^2)$
The first term is due to the creation of the two input matrices and the output one. The second term is due to the creation of matrices C_{ij}. The third term is due to to the recursion calls [2].

References

[1] Andrea Gonzato: Python implementation of matrix multiplication,
`https://github.com/AndreaGonzato/Stassen_Algorithmic_Design/blob/main/matrix.py`

[2] Gary Miller: Introduction and Strassen's Algorithm, Section 2.5 Running Strassen's Algorithm using only $O(n^2)$ space
`http://www.cs.cmu.edu/afs/cs/academic/class/15750-s17/ScribeNotes/lecture1.pdf`